Waves, Displacement, Reflections

Arun Rao
CS 594

Goals

- Simple looking waves
- Waves displace and change orientation of things on top
- Reflections on surface
- Partially see through

How do we create the Waves?

- Large Mesh
  - 120 x 120 quads of size 1
- Vertex Shader
  - use sine functions
  - needs to change based on time

Waves.vert ver. 1

```
uniform float   Time;

const float     xdiv = 10.0; // scalar applied to gl_Vertex.x
const float     zdiv = 8.0;  // scalar applied to gl_Vertex.y
const float     magx = 1.6;  // magnitude of wave coming in x
const float     magz = 1.0;  // magnitude of wave coming in z

void main(void)
{
    float h1, h2;

    // determine current position
    h1 = sin((gl_Vertex.z / zdiv) + Time) * magz;
    h2 = sin((gl_Vertex.x / xdiv) + Time) * magx;
    h1 = h1 + h2;

    // apply position
    gl_Position = gl_ModelViewProjectionMatrix *
        vec4(gl_Vertex.x, gl_Vertex.y + h1, gl_Vertex.z,1);
}
```

What about Lighting?

- Assuming we've setup lighting correctly
  - light position, diffuse and ambient material setup
  - We'll do lighting per-fragment
- Need to calculate surface normal
  - Take partial derivative of equation in x and z directions
  - d/dx(sin(u)) = cos(u) * d/dx(u)
  - Take cross product of two vectors

Code?

- Calculate normal

```
varying vec3 normal
...
...
vec3 v1; // along +z axis
vec3 v2; // along +x axis

// get our two slopes in xy and zy
// d( sin(u) ) = cos(u)d(u)
v1 = vec3(  0.0,
            cos((gl_Vertex.z / zdiv) + Time) * magz / zdiv,
            1.0);
v2 = vec3(  1.0,
            cos((gl_Vertex.x / xdiv) + Time) * magx / xdiv,
            0.0);

// find our normal, it will definitely point up!
normal = normalize(cross(v1,v2));
```

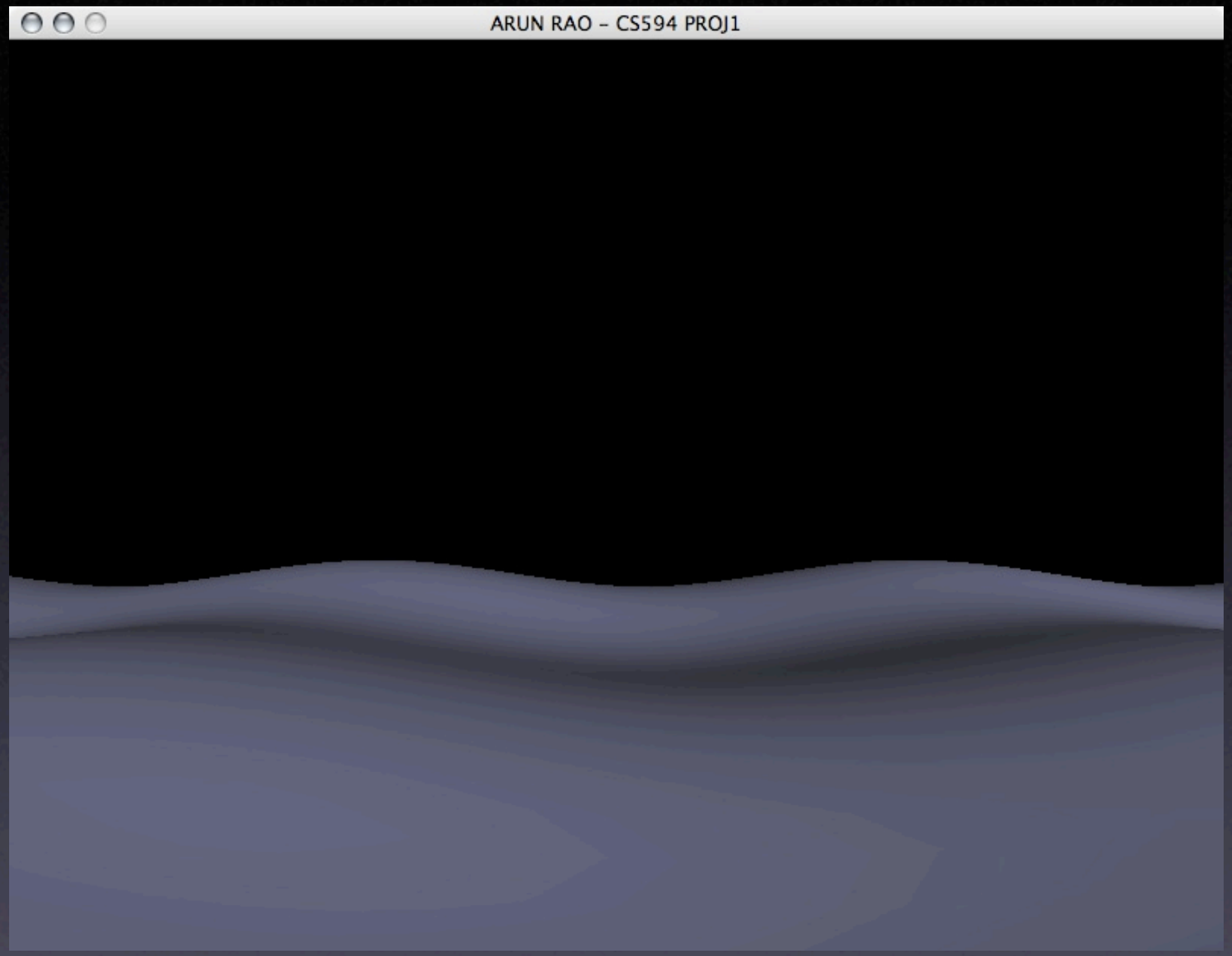
Code?

```
varying vec3 lightDir, halfVector;
varying vec4 diffuse, ambient;

...
    normal = normalize(gl_NormalMatrix * normal);
    lightDir = normalize(vec3(gl_LightSource[1].position));
    halfVector = normalize(gl_LightSource[1].halfVector.xyz);

    diffuse = gl_FrontMaterial.diffuse * gl_LightSource[1].diffuse;
    ambient = gl_FrontMaterial.ambient * gl_LightSource[1].ambient;
    ambient += gl_LightModel.ambient * gl_FrontMaterial.ambient;
```

Code?

- Fragment Shader

```
void main()
    {
        vec3 n,halfV;
        float NdotL,NdotHV;

        /* The ambient term will always be present */
        vec4 color = ambient;

        /* a fragment shader can't write a varying variable, hence we need
        a new variable to store the normalized interpolated normal */
        n = normalize(normal);

        /* compute the dot product between normal and ldir */
        NdotL = max(dot(n,lightDir),0.0);

        if (NdotL > 0.0) {
            color += diffuse * NdotL;
            halfV = normalize(halfVector);
            NdotHV = max(dot(n,halfV),0.0);
            color += gl_FrontMaterial.specular *
                     gl_LightSource[0].specular *
                     pow(NdotHV, gl_FrontMaterial.shininess);
        }

        gl_FragColor = color;
}
```



Awesome!!!

How do we move our objects?

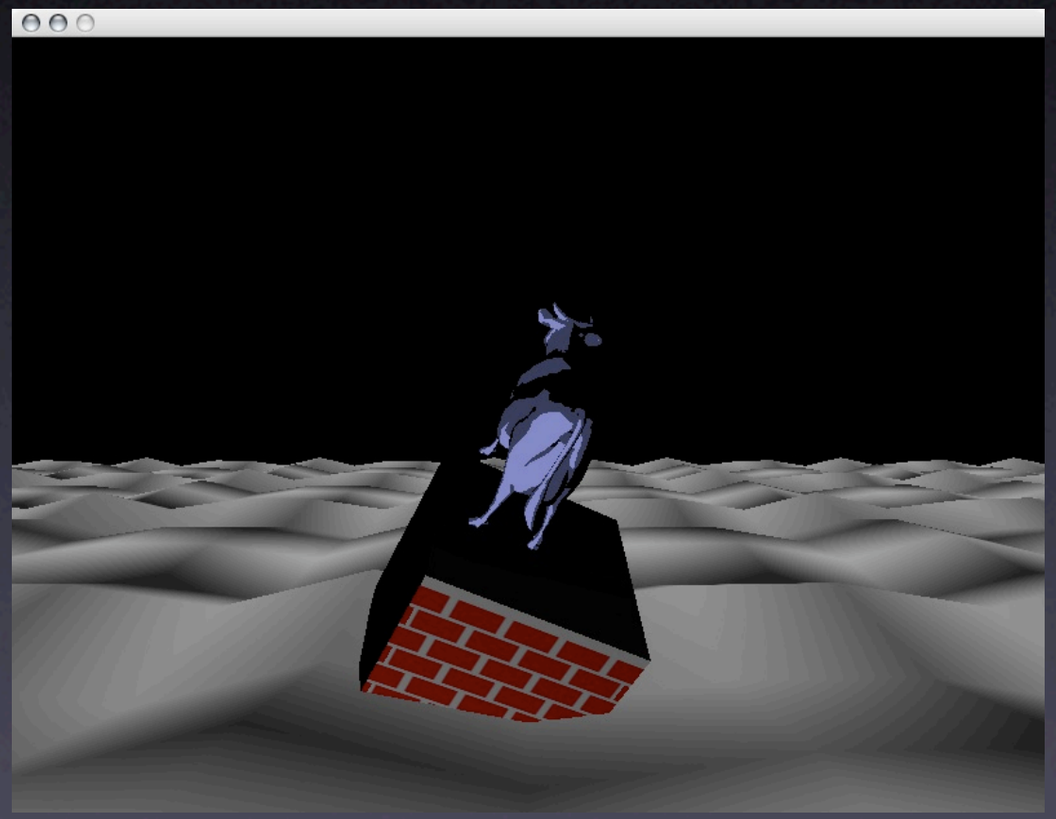
- Incorporate same code as above into our other shaders as a function
- Pick a point to apply displacement and change of normals
- I cheat and use fixed point because of later on

Loosing direction!!

- We need an object rotating in place
  - Push Matrix
  - Position
  - Rotate
  - Apply Displacement & Orientation via Vertex Shader
  - Pop Matrix
  - PROBLEMS!!!!!

Order Matters!!

- Wave function applied to objects cause:
  - normal matrix to be wrong
  - lighting to get screwed up
  - doesn't look like wading through water
- Hacky fix by doing transforms in shader and rebuilding normal matrix per-vertex!!!



What do we do in our vertex shaders?

- Copy the ModelView Matrix & Normal Matrix (mvmat & nmat)
- Run the Wave function and get our vertical displacment, normal to surface and basis matrix for rotation
- Perform translate then rotate on copy of ModelView Matrix
- Apply spin about RELATIVE y-axis to ModelView Matrix
- Rebuild Normal Matrix
- calculate varying normal to pass to fragment shader
- gl_Position = gl_ProjectionMatrix * mvmat * v

```
vec3 testPosition = vec3( 0.0, 0.0, -10 );
mat4 wave_rot_mat;
...
void wave_func( inout vec3 tp, inout vec3 tn)
{
    vec3 v1, v2, pos1, pos2;
    float h1, h2;

    // determine current height displacement
    h1 = sin((testPosition.z / zdiv) + Time) * magz;
    h2 = sin((testPosition.x / xdiv) + Time) * magx;
    tp = vec3( 0.0, h1 + h2, 0.0); // only take y component

    // get our two slopes in xy and zy
    // d/dx( sin(u) ) = cos(u)d/dx(u)
    // v1 with respect to z
    // v2 with respect to x
    v1 = vec3(  0.0,
                cos((testPosition.z / zdiv) + Time) * magz / zdiv,
                1.0);
    v2 = vec3(  1.0,
                cos((testPosition.x / xdiv) + Time) * magx / xdiv,
                0.0);

    // find our normal, should be in positive y direction
     v1 = normalize(v1);
     v2 = normalize(v2);
     tn = normalize(cross(v1,v2));

    // should have our basis vectors!! v1, v2, tn... now build a matrix
    wave_rot_mat = identity;
    wave_rot_mat[0] = vec4( v2, 0.0);
    wave_rot_mat[1] = vec4( tn, 0.0);
    wave_rot_mat[2] = vec4( v1, 0.0);
}
```

```
void main(void) {
    vec3 tp = vec3(0.0,0.0,0.0);
    vec3 tn = vec3(0.0,0.0,0.0);
    mat4 mvmat = gl_ModelViewMatrix;
    mat3 nmat = gl_NormalMatrix;

    // apply wave displacement
    wave_func(tp,tn);

    // translation
    mat4 tempmat = identity;
    tempmat[3][0] = testPosition.x;
    tempmat[3][1] = tp.y;
    tempmat[3][2] = testPosition.z;
    mvmat *= tempmat;

    // apply rotation due to normal at wave
    mvmat *= wave_rot_mat;
    ...
```

```
uniform float Spin;  // a rotation value sent in
...
mat3 nmat = gl_NormalMatrix;
...
// did our translate and rotate due to wave
...
// apply rotation about y-axis
 tempmat = identity;
 tempmat[0][0] = cos(Spin);
 tempmat[2][0] = sin(Spin);
 tempmat[0][2] = -sin(Spin);
 tempmat[2][2] = cos(Spin);
 tempmat[3][1] = CubeScale;
 mvmat *= tempmat;

 // rebuild normal matrix
 build_norm_mat(mvmat,nmat);

 // we happen to know the scale of the cube below us,
// let's reuse tp
 tp = gl_Vertex.xyz;
 normal = normalize( nmat * gl_Normal );
 gl_Position = gl_ProjectionMatrix * mvmat * vec4(tp,1);
```

How do we build the normal matrix?

- Need the upper-left 3x3 of ModelView Matrix
- Invert
  - need determinant
- Transpose
- Remember $(A^{-1})^T = (A^T)^{-1}$

Inverse of 3x3 Matrix

$$\mathbf{A}^{-1} = \frac{1}{\det \mathbf{A}} \begin{pmatrix} a_{22}a_{33} - a_{23}a_{32} & a_{13}a_{32} - a_{12}a_{33} & a_{12}a_{23} - a_{13}a_{22} \\ a_{23}a_{31} - a_{21}a_{33} & a_{11}a_{33} - a_{13}a_{31} & a_{13}a_{21} - a_{11}a_{23} \\ a_{21}a_{32} - a_{22}a_{31} & a_{12}a_{31} - a_{11}a_{32} & a_{11}a_{22} - a_{12}a_{21} \end{pmatrix}$$

Transpose of 3x3 Matrix

$$\text{Transpose of } \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^T$$

$$\Rightarrow \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

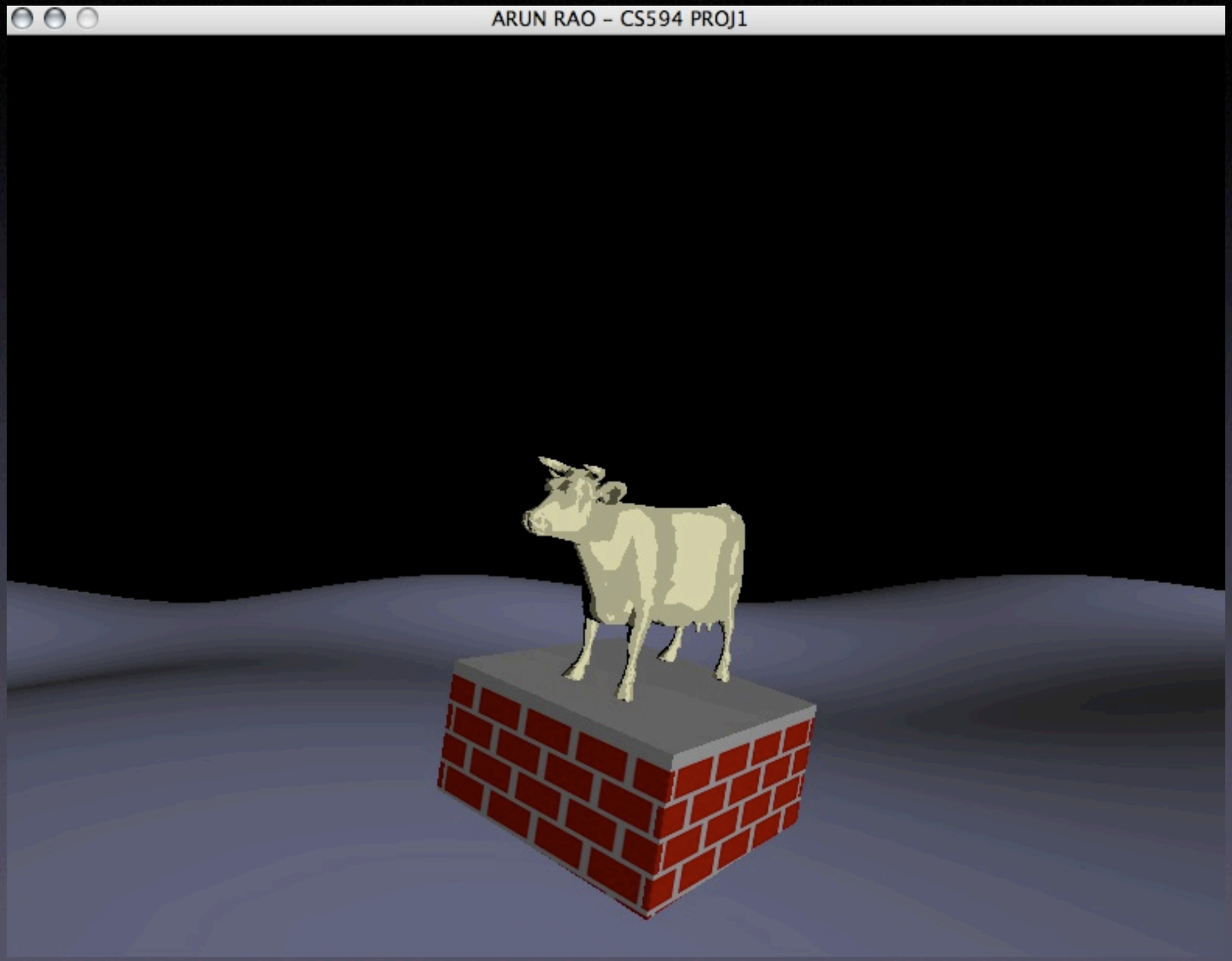
Do them both at the same time!

```
mat3  tnmat;
float d;
for( int i = 0; i < 3; i++)
    for( int k = 0; k < 3; k++)
        nmat[i][k] = mvmat[i][k];
d = determinant(nmat);

// take determinant and transpose at same time
tnmat[0][0] =
    determinant(mat2(nmat[1][1], nmat[1][2], nmat[2][1], nmat[2][2]));
tnmat[1][0] =
    determinant(mat2(nmat[0][2], nmat[0][1], nmat[2][2], nmat[2][1]));
tnmat[2][0] =
    determinant(mat2(nmat[0][1], nmat[0][2], nmat[1][1], nmat[1][2]));
tnmat[0][1] =
    determinant(mat2(nmat[1][2], nmat[1][0], nmat[2][2], nmat[2][1]));
tnmat[1][1] =
    determinant(mat2(nmat[0][0], nmat[0][2], nmat[2][1], nmat[2][2]));
tnmat[2][1] =
    determinant(mat2(nmat[1][2], nmat[0][0], nmat[1][2], nmat[1][0]));
tnmat[0][2] =
    determinant(mat2(nmat[1][0], nmat[1][1], nmat[2][0], nmat[2][1]));
tnmat[1][2] =
    determinant(mat2(nmat[0][1], nmat[0][0], nmat[2][1], nmat[2][0]));
tnmat[2][2] =
    determinant(mat2(nmat[0][0], nmat[0][1], nmat[1][0], nmat[1][1]));

nmat = tnmat / d;
```

Determinant Functions

```
float determinant(mat3 m)
{
    float t = 0.0;
    t = m[0][0] * ((m[1][1] * m[2][2]) - (m[2][1] * m[1][2]));
    t+= m[0][1] * ((m[1][2] * m[2][0]) - (m[2][2] * m[1][0]));
    t+= m[0][2] * ((m[1][0] * m[2][1]) - (m[2][0] * m[1][1]));

    return t;
}

float determinant(mat2 m)
{
    return ((m[0][0] * m[1][1]) - (m[1][0] * m[0][1]));
}
```

More Awesome!!!



Still need reflections!

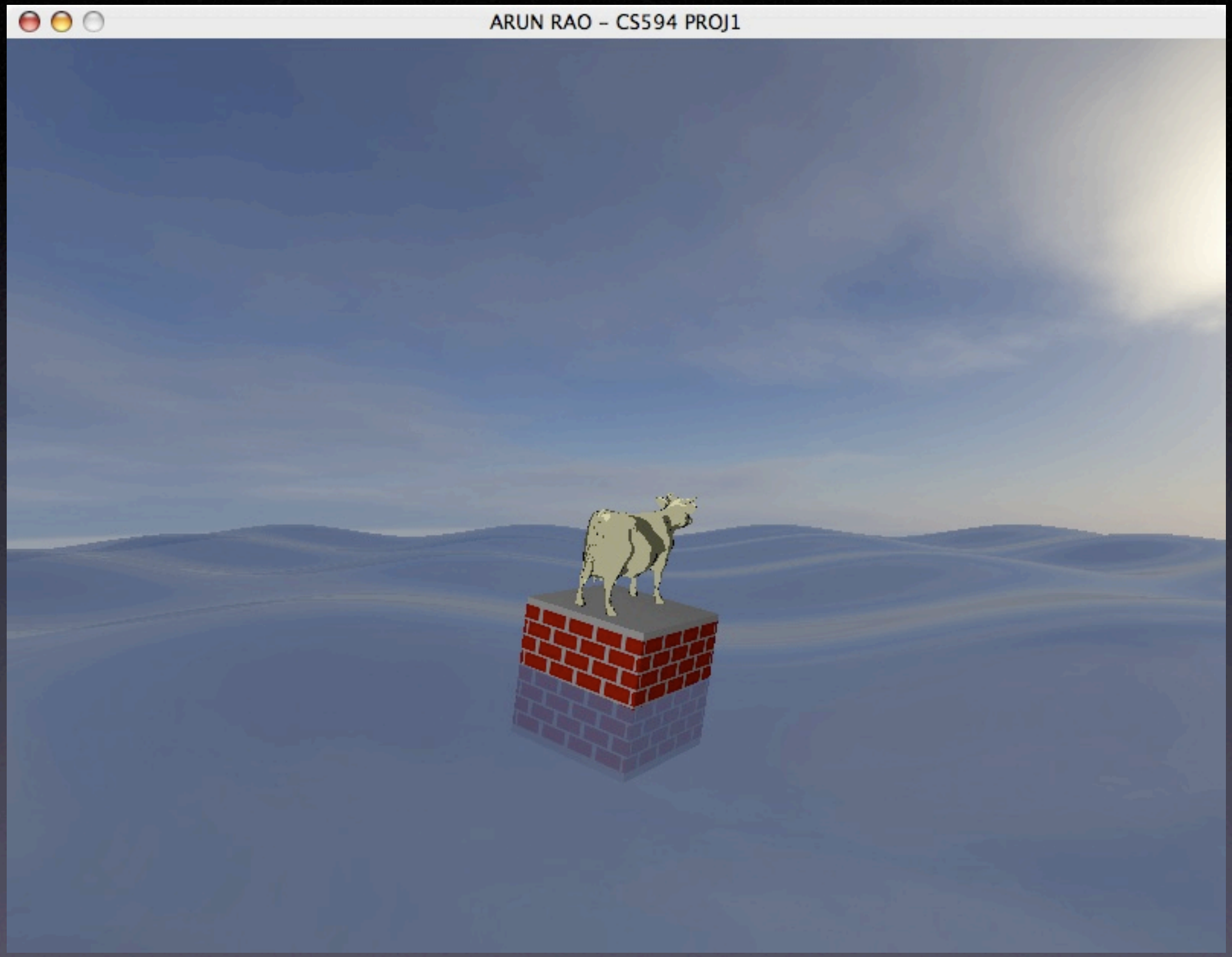
- Found a cubemap online
- Did the GL_TEXTURE_CUBE_MAP bindings
- Access of bound cubemap in shader by
  - texture unit
  - not texture object id!
- Need eye direction and normal of surface
- Perform texture lookup in frag. shader

Need The Eye Direction Vector from Vertex Shader

```
varying vec3 eyeDir;
...
    float h1, h2;
    ...
    // determine current position
    h1 = sin((gl_Vertex.z / zdiv) + Time) * magz;
    h2 = sin((gl_Vertex.x / xdiv) + Time) * magx;
    h1 = h1 + h2;
    vec4 p = vec4( gl_Vertex.x, gl_Vertex.y + h1, gl_Vertex.z, 1);
    ...
    eyeDir = vec3( gl_ModelViewMatrix * p);
    ...
```

In the Frag Shader

```
uniform samplerCube     CubeMapTex;

varying vec3 normal, lightDir, halfVector, eyeDir;
varying vec4 diffuse,ambient;

void main(void)
{
    float NdotL, NdotHV;
    vec4 color = ambient;
    vec3 n = normalize(normal);
    NdotL = max(dot(n,lightDir),0.0);
    if( NdotL > 0.0 )
    {
        color += diffuse * NdotL;
        NdotHV = max( dot( n, normalize(halfVector)), 0.0);
        color += gl_FrontMaterial.specular * gl_LightSource[1].specular *
            pow(NdotHV, gl_FrontMaterial.shininess);

    }

    // our cube map forces us to make the reflectDir negative...
    // mainly cause of our texture resources.... anyway :)
    vec3 reflectDir = -reflect( eyeDir, normal);
    vec4 texturecolor = textureCube( CubeMapTex, normalize(reflectDir));
    gl_FragColor = vec4( vec3(mix( color, texturecolor, .6)) , .8);
}
```


ARUN RAO - CS594 PROJ1